PASCAL 64

Der PASCAL-Compiler für den

COMMODORE 64

EIN DATA BECKER PROGRAMM

PASCAL 64

Der PASCAL-Compiler für den

COMMODORE 64

EIN DATA BECKER PROGRAMM

Wichtiger Hinweis

Das vorliegende Handbuch und das dazugehörige Programm wurden vom Autor mit größter Sorgfalt erarbeitet und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf Programmfehler oder fehlerhafte Angaben im Handbuch zurückgehen, übernommen werden kann. Für die schriftliche Mitteilung eventueller Fehler sind wir jederzeit dankbar.

Copyright (C) 1983 DATA BECKER
Merowingerstr. 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil des Handbuches und des dazugehörigen Programms darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

A C H T U N G

DIESES PROGRAMM IST GEGEN KOPIEREN GESCHÜTZT

Natürlich ist kein Kopierschutz 100%ig sicher, aber wir werden mit allen uns zur Verfügung stehenden rechtlichen Mitteln versuchen, solchen Software-Piraten das Handwerk zu legen.

+++ KOPIERVERSUCHE KÖNNEN DIE DISKETTE ZERSTÖREN! +

Sollte aus irgendeinem Grund Ihre Diskette beschädigt oder gar zerstört werden, so senden Sie uns bitte Ihre Originaldiskette mit einem Verrechnungsscheck über DM 10.- an uns zurück. Sie erhalten dann postwendend eine neue Diskette.

INHALTSVERZEICHNIS

1. KAPITEL

1.1 Die Funktionsweise des PASCAL 64 Compilers 1

2. KAPITEL

2.1 Grundorganisation eines PASCAL-Programmes 3
2.2 Definition eines Blockes 3
2.3 Allgemeine Syntax 4
2.4 Kommentare 4

3. KAPITEL

3.1 Variablentypen und Ausdruecke 5
3.2 Rechenverknuepfungen 6
3.3 Variablendeklaration 7
3.4 Konstantendeklarationen 8
3.5 Arithmetische Ausdruecke 8
3.6 Die Zuweisung in PASCAL 9

4. KAPITEL

4.1 Ein- und Ausgabe 11
4.2 Weiterfuehrende Ein- und Ausgabemoeglichkeiten 12
4.3 Der Filestatus 14

5. KAPITEL

5.1 Kontrollstrukturen 15
5.2 Wiederholungsanweisungen (Schleifen)
5.2.1 Die REAPEAT-Anweisung 15
5.2.2 Die WHILE-Anweisung 16

5.2.3 Die FOR-Anweisung 17

5.3 Entscheidungsanweisungen

5.3.1 Die IF-Anweisung 18

5.3.2 Die CASE-Anweisung 19

6. KAPITEL

6.1 Unterprogramme und Funktionen

6.1.1 Unterprogrammdefinition 22

6.1.2 Unterprogrammaufruf 23

6.1.3 Definition von Funktionen 24

6.2 Rekursionen 25

7. KAPITEL

7.1 Grafik-Befehle

7.1.1 Einschalten der Grafik 27

7.1.2 Loeschen der Grafik 27

7.1.3 Setzen eines Punktes 27

7.1.4 Loeschen eines Punktes 28

7.1.5 Der SPRITE-Befehl 28

7.2 Der POKE-Befehl 29

7.3 Der SYS-Befehl 29

7.4 Der INTEGER; Befehl 30

7.5 Abschalten des Ganzzahlmodus (REAL) 30

7.6 Fehlermeldungen des uebersetzen Programmes 31

8. KAPITEL

8.1 Zeichnen einer Linie 33

8.2 Optimieren einer vierfarbigen Grafik 33

8.3 MULTIPLOT 34

9. KAPITEL

9.1 Einfuehrung in das strukturierte Programmieren 35

10.KAPITEL

10.1 PASCAL 64 Kurzübersicht

10.1.1 Datentypen 38

10.1.2 Deklarationen 38

10.1.3 Unterprogramme 38

10.1.4 Strukturen 38

10.1.5 Ein- und Ausgabebefehle 39

10.1.6 Sonstige Befehle 40

10.1.7 Zuordnungen 41

10.1.8 Rechenoperationen 41

10.1.9 Funktionen 41

1. KAPITEL

1.1 Die Funktionsweise des Pascal 64 Compilers

"Compiler", 8 oder "40x", 8

-1- Laden Sie den Compiler mit Load ~~"Pascal 64 Compiler"~~,8

-2- Zum Schreiben des Pascal-Programmes stehen Ihnen nun die Zeilennummern von 1 bis 9999 zur Verfuegung. Alle anderen Zeilen werden vom Compiler belegt und duerfen deshalb nicht benutzt werden. Das Programm kann nun genauso geschrieben werden, wie Sie dies von BASIC bereits gewohnt sind.
Achtung: In Pascal kann es vorkommen, dass am Anfang einer Zeile eine Zahl stehen muss (CASE-Befehl). Der BASIC-Editor sieht diese Zahl als Ergaenzung der Zeilennummer und generiert eine falsche Zeile. Um dieses zu verhindern, muss ein geshiftetes Leerzeichen die Zahl von der Zeilennummer trennen. Dieses muss auch vor jeder Korrektur der entsprechenden Zeile geschehen. Die BASIC-Befehle REM und DATA beeinflussen den Editor. Sie duerfen deshalb in keiner Form im Programm erscheinen, auch nicht als Teil eines Variablennamens.

-3- Nach Fertigstellung des Programmes wird der Compiler mit RUN gestartet. Er fragt nun nach dem Programmnamen, dem verwendeten Massenspeicher (C=Cassette, D=Diskette) und nach der Verschachtelungstiefe des Programmes (Schleifenmaximalverschachtelung). Es kann auch ein Wert groesser dem tatsaechlichen Wert angegeben werden.

-4- Der Compiler schreibt nun das Maschinenprogramm auf den angegebenen Massenspeicher und zeigt in der ersten Bildschirmzeile die bearbeitete Zeile an.

-5- Eine nicht uebersetzbare Zeile meldet der Compiler meistens mit ?COMPILE ERROR IN XXXX ;sollte eine andere Meldung ausgegeben werden, koennen Sie die fehlerhafte Zeile mit GOTO39000 ermitteln. Das Programm kann nun mit LIST-9999 gelistet und korrigiert werden. Danach ist bei Schritt 3

fortzufahren.

-6- Nachdem das fehlerlose Programm uebersetzt wurde, meldet der Compiler das Programmende, welches notiert werden muss. Das Pascal-Programm sollte nun samt Compiler mit SAVE"Name",1oder8 abgespeichert werden, falls man es zur Verbesserung noch einmal benoetigt.

-7- Nun den Pascal-Loader laden und mit GOTO100 starten. Der Loader fragt nach dem Programmende (Punkt 6), dem Programmnamen und laedt dann das erzeugte Maschinenprogramm.

-8- Der Loader meldet sich mit READY und das fertige Programm kann mit RUN gestartet, mit SAVE gespeichert und mit LOAD wieder geladen werden. Bei Benutzung der Grafik wird vom Compiler entsprechend viel Speicherplatz im Maschinenprogramm reserviert. Wollen Sie diesen Bereich nicht abspeichern, geben Sie vor dem SAVE den Befehl SYS 3707 ein. Um ein derart gerafftes Programm zu entraffen und zu starten, benutzen Sie SYS 3748. Ein Programm darf nur einfach gerafft oder entrafft werden und nur bei Benutzung der Grafik (Kapitel 7).

-9- Um die Bedienung des Compilers zu lernen, ist es empfehlenswert, erst die folgenden Kapitel zu lesen und dann eines der Beispielprogramme einzugeben.

2. KAPITEL

2.1 Grundorganisation eines Pascal-Programmes

-1- Variablendeklarationen

Hier werden alle Variablen, Konstanten und Felder des Hauptprogrammes definiert und ihnen ein Variablentyp zugeordnet.

-2- Unterprogrammdeklarationen

In diesem Teil werden die vom Hauptprogramm oder von anderen Unterprogrammen benutzten Unterprogramme und Funktionen definiert.

-3- Hauptprogramm

Der Hauptprogrammblock beginnt mit BEGIN und wird mit END. geschlossen.

Diese Teile muessen in der obigen Reihenfolge stehen, wobei Unterprogrammbloecke wiederum aus Variablen- und Programmteil bestehen. Das Hauptprogramm ist der zuerst ausgefuehrte Programmteil. Nach dem END. des Hauptprogrammes wird die Kontrolle wieder an den BASIC-Interpreter uebergeben.

2.2 Definition eines Blockes:

Ein Block ist ein Programmteil, der in BEGIN und END; eingeschlossen ist und aneinandergereihte Befehle, Unterprogrammaufruefe und Strukturen (Schleifen, etc.) enthaelt. Innerhalb von Strukturen stehen dann widerum ein oder mehrere Bloecke. Ein Block, der nur aus einem Befehl oder nur einer Struktur besteht, braucht nicht in BEGIN und END; eingeschlossen werden.

2.3 Allgemeine Syntax:

Ein Einzelbefehl wird durch ein Semikolon und ein darauf folgendes Leerzeichen abgeschlossen. Nach einem Befehlswort muss immer ein Leerzeichen folgen.
Beispiel:

```
READLN (Variable1,Variable2); WRITELN (Variable1,Variable2);
```

Leerzeichen sollten immer so gesetzt werden, wie es bei den jeweiligen Befehlsbeispielen demonstriert ist. Listenelemente bei Variablennamen, Parametern oder Ausdruecken werden durch Kommata getrennt und nicht durch Leerzeichen (z.B. die Auflistung der Variablennamen bei der Variablendeklaration).

2.4 Kommentare

Um einen Kommentar in ein Pascal Programm einzufuegen, wird dieser folgendermassen gekennzeichnet:

```
(* Kommentar *)
```

Der Compiler beachtet den Kommentar nicht.
Naeheres zu oben Gesagtem wird in den jeweiligen Kapiteln erklaert.

3. Kapitel

3.1 Variablentypen und Ausdruecke

In Pascal 64 sind drei Variablentypen moeglich:

Ganze Zahlen werden in Pascal mit INTEGER bezeichnet und duerfen einen Wert zwischen -32768 und 32767 einnehmen. Nachkommastellen werden von der entsprechenden Variablen nicht aufgenommen. INTEGER-Variablen belegen nur wenig Speicher (2 Bytes) und werden schnell verarbeitet, Variablenfelder sollten deshalb moeglichst vom Typ INTEGER sein.

Reelle Zahlen werden mit REAL bezeichnet und haben dieselben Wertebereiche und Stellenanzahl wie BASIC-Gleitkomma-Variablen. Eine Gleitkommazahl kann maximal +- 1.70141183E+38 erreichen. Die Rechengenauigkeit ist auf 9 Stellen beschraenkt. Jede Gleitkommazahl belegt fuenf Bytes. Im Gegensatz zu BASIC ist es in Pascal nicht erlaubt als erstes Zeichen einer Zahl den Dezimalpunkt zu benutzen, stattdessen muss in solchen Faellen eine Null vor den Dezimalpunkt gestellt werden:

.00315 wird zu 0.00315

Alphanumerische Zeichen werden als CHAR bezeichnet und duerfen alle auf dem C-64 darstellbaren Zeichen enthalten, einschliesslich der Kontrollfunktionen und der Cursorsteuerung. Alphanumerische Zeichen werden in Anfuehrunszeichen eingeschlossen. Der Typ CHAR kann keine Zeichenketten aufnehmen, dies muss ueber Variablenfelder vom Typ CHAR geschehen.

Die Umwandlung von INTEGER in REAL und umgekehrt wird automatisch vollzogen, falls dies bei einer Berechnung noetig sein sollte und moeglich ist. Bei der Umwandlung von REAL in INTEGER fallen natuerlich die Nachkommastellen weg, dies ist z.B. bei den Operationen DIV und MOD der Fall und bei der

Uebernahme von Gleitkommaergebnissen in eine INTEGER-Variable.

3.2 Rechenverknuepfungen

Folgende Rechenverknuepfungen sind in Pascal moeglich:

+, -, *, / ;Diese Operationen arbeiten wie in BASIC und verknuepfen REAL oder INTEGER.
DIV ;Diese Operation arbeitet wie eine Division, liefert aber einen Wert vom Typ INTEGER und wird deshalb schneller ausgefuehrt.
MOD ;Diese Verknuepfung arbeitet wie DIV. Es wird aber der mathematische Rest der Division als Ergebnis geliefert.
AND, OR ;Diese Operationen arbeiten widerum wie in BASIC, liefern einen Wert vom Typ INTEGER und koennen als logische oder binaere Operationen benutzt werden.
Die Vergleichsoperationen >, <, =, >=, <=, <> arbeiten auch wie in BASIC und sind fuer alle drei Datentypen zulaessig. Das Ergebnis dieser Operationen ist entweder -1 = wahr oder 0 = unwahr. Zwei derartige Ergebnisse koennen mit AND oder OR verknuepft werden und liefern wieder ein Ergebnis vom Typ wahr oder unwahr. Von dieser Abfrage machen Schleifen (REPEAT, WHILE) und IF Gebrauch. Ein solches Ergebniss laesst sich aber auch einer Variablen zuordnen und erst spaeter mit IF testen. Die Funktion NOT(X) kehrt ein solches Ergebnis in sein Gegenteil um.

Ausserdem gibt es in Pascal folgende vordefinierte Funktionen:

SIN(X), COS(X), TAN(X), EXP(X), ABS(X), PEEK(X), NOT(X) ;entsprechen den gleichnamigen BASIC-Funktionen. Das Argument kann vom Typ REAL oder INTEGER sein.
TRUNC(X) ;liefert den ganzzahligen Anteil einer Fliesskommazahl und entspricht in BASIC dem INT.
SQR(X) ;quadriert eine Zahl.
SQRT(X) ;ermittelt die Quadratwurzel einer Zahl und entspricht der BASIC-Funktion SQR.
LN(X) ;entspricht der BASIC-Funktion LOG.

ARCTAN(X) ;Umkehrung von TAN, entspricht in BASIC dem ATN.
ORD(X) ;wandelt ein Zeichen in den entsprechenden Zahlencode. Entspricht in BASIC der ASC-Funktion.
CHR(X) ;wandelt eine Zahl in ein Zeichen und ist somit die Umkehrung von ORD ;BASIC-Funktion CHR$.

Operanden von Funktionen muessen in Klammern eingeschlossen werden, dies gilt auch fuer die NOT-Funktion.
Sicherlich vermissen Sie einige mathematische Funktionen, dies ist aber kein prinzipielles Problem. Sie lassen sich mit Hilfe einer Formelsammlung leicht erstellen:

```
X hoch Y  : EXP(Y*LN(X))
COT(X)    : 1/TAN(X)
ARCSIN(X) : ARCTAN(X/SQRT(1-X*X))
COSH(X)   : (EXP(X)+EXP(-X))/2
LOG10(X)  : LN(X)/LN(10)
```

In Pascal ist es moeglich beliebige weitere Funktionen zu definieren. Darauf wird in Kapitel 6 eingegangen.

Alle drei Datentypen koennen als Variable, eindimensionales Variablenfeld, Konstante oder direkte Zahl in mathematischen Ausdruecken erscheinen.
Alle in einem Programm oder Unterprogramm benutzten Variablen muessen vor dem Programmbeginn bereits deklariert sein. (siehe Kapitel 2)

3.3 Variablendeklaration

Der Befehl VAR teilt dem Compiler mit, dass nun Variablennamen folgen. Diese werden durch Kommata voneinander getrennt, nach einem Doppelpunkt steht der Typ dieser Variablen, und ein Semikolon gefolgt von einem Leerzeichen trennt sie von den naechsten Variablen.

Beispiel:

```
VAR A,B,TEST:REAL; X,ZAEHLER:INTEGER; A:REAL; M,N:CHAR;
```

Eine deklarierte Variable hat noch keinen definierten Ausgangszustand, auch nicht den Wert Null. Ihr muss erst vom Programm ein Wert zugeordnet werden, bevor auf ihren Inhalt zugegriffen wird, sonst enthaelt sie einen zufaelligen Wert. Die Laenge der Variablennamen und ihre Anzahl ist beliebig. Ein Variablenname darf im ganzen Programm nur einmal vergeben werden und darf keine Pascal-Verknuepfungsbefehle enthalten. Der Name COLOR enthaelt z.B. OR und ist damit nicht erlaubt. Zum Definieren von Variablenfeldern wird der Variablentyp einfach durch ein ARRAY(0..Zahl) OF Variablentyp; ersetzt:

```
VAR A,B,TEST:ARRAY(0..1000) OF REAL;
```

Es werden die Felder A, B und TEST definiert, die alle von 0 bis 1000 laufen. Dies entspricht dem BASIC-Befehl DIM. Normalerweise werden in Pascal eckige Klammern benutzt, um den Index eines Feldes einzuklammern. Aus technischen Gruenden stehen in dieser Anleitung nur runde Klammern. In Pascal 64 ist dagegen beides erlaubt.

3.4 Konstantendeklaration

Der Befehl CONST wird vor die Auflistung der Konstanten gestellt und dann den Konstanten mit Name=Wert; Werte zugeordnet. Der Wert einer Konstanten bleibt ueber das ganze Programm gleich.

```
CONST PI=3.14159265; TRUE=-1; FALSE=0; KLAMMER="(";
```

Die Datentypen werden vom Compiler automatisch hinzugefuegt. Fuer Konstantennamen gilt dasselbe wie fuer Variablennamen.

3.5 Arithmetische Ausdruecke

Der Aufbau eines Ausdrucks (Formel) geschieht in Pascal wie in BASIC, d.h. Aneinanderreihung von Verknuepfungen, algebraische Rechenhierarchie, Benutzung aller Variablen- und Datentypen und uneingeschraenkter Einsatz von Klammerebenen ist in Pascal voll moeglich. Somit ist ein vollstaendiges

algebraisches Operationssystem implementiert. Es sind dabei folgende Ausnahmen zu beachten:

Innerhalb eines Ausdrucks bzw. einer Formel duerfen keine Leerzeichen benutzt werden.

Der Index von Feldvariablen ist in eckige Klammern zu setzen und kann selbst wieder ein Ausdruck sein.

Selbstdefinierte Funktionen koennen wie eingebaute Funktionen benutzt werden, Funktionsargumente koennen selber wieder Ausdruecke sein.

Die Negation einer Variablen durch das Voranstellen eines Vorzeichens ist nicht erlaubt:

B:= -B; muss heissen B:= B*-1;

Die Rechenhierarchie in Pascal ist geringfuegig anders als in BASIC:

- Die Klammern besitzen den hoechsten Vorrang. Dies gilt natuerlich auch fuer Funktionsparameter, die in Klammern eingeschlossen werden muessen.
- Dann werden die Operationen *, /, DIV, MOD und AND ausgefuehrt.
- Darunter liegen +, - und OR.
- Zuletzt werden die Vergleichsoperationen (<, >, =, <>, <=, >=) ausgefuehrt.

Die logischen Operationen AND und OR liegen in dieser Hierarchie etwas unguenstig. Diese ist aber in Pascal Standard und wurde deshalb auch in Pascal 64 uebernommen. Es empfiehlt sich deshalb bei entsprechenden Ausdruecken immer Klammern zu setzen, um Fehler zu vermeiden.

3.6 Die Zuweisung in Pascal

Um einer Variablen einen Wert zuzuweisen benutzt man folgende Anweisung:

Variable:= Ausdruck;

oder bei Variablenfeldern:

Variable(Ausduck1):= Ausdruck2;

Der Variablen auf der linken Seite wird das Ergebnis des Ausdrucks auf der rechten Seite zugeordnet. Diese Zuweisung arbeitet analog zu der in BASIC. Es wird in Pascal nur nach dem Gleichheitszeichen ein Doppelpunkt gesetzt, um die Zuweisung von einem Vergleich zu unterscheiden.
Beispiel:

FELD(3+4):= SIN(8);

Das siebte Feldelement enthaelt nun den Sinus von 8, vorausgesetzt das Feld ist vom Typ REAL.

4. KAPITEL

4.1 Ein- und Ausgabe

Um eine Zahl oder ein Zeichen von der Tastatur einzulesen, benutzt man in Pascal den READ-Befehl:

```
READ (Variable1,Variable2,...,VariableN);
```

Dieser Befehl bewirkt, dass die durch Kommata getrennten Variablen eingelesen werden. Bei der Eingabe sind die entsprechenden Zahlenwerte durch Leerzeichen oder die Return-Taste zu trennen. Natuerlich koennen auch Feldvariablen eingelesen werden. Sollte die Variable vom Typ CHAR sein, wird nur ein Zeichen eingelesen.
Anstelle des READ-Befehls kann man auch den READLN-Befehl benutzen, welcher nach Beendigung einer Eingabe ein Return erwartet:

```
READLN (X,Y);
```

Wuerde jetzt folgende Eingabe gemacht werden:

```
12 32.5 4 'RETURN'
```

dann wuerden die ersten beiden Zahlen in die Variablen X und Y geschrieben werden, die Zahl 4 wuerde aber ignoriert werden. Der READ Befehl wuerde dagegen die Zahl 4 fuer die naechste Eingabe stehenlassen.

Um Ergebnisse auszugeben wird in Pascal der WRITE-Befehl benutzt:

```
WRITE (Variable,Formel,"Text",...);
```

Es ist mit dem WRITE-Befehl moeglich, das Ergebnis jeder beliebigen Formel auszugeben. Ausserdem koennen Texte ausgegeben werden, die in Anfuehrunszeichen zu setzen sind. Texte duerfen alle Zeichen enthalten, die zwischen Anfuehrungszeichen darstellbar sind (z.B. die Cursorzeichen).

Der WRITELN Befehl entspricht dem WRITE Befehl, setzt aber nach der Ausgabe noch einen 'RETURN'-Code.Um nur ein 'RETURN' auszugeben, schreibt man den WRITELN-Befehl ohne Parameter:

```
WRITELN;
```

Entsprechende BASIC-Befehle sind:

```
READ : INPUT
WRITELN : PRINT
WRITE : PRINT ... ;
```

Bei der Eingabe von Zahlen muessen diese durch Leerzeichen getrennt werden und nicht durch Kommata wie in BASIC.

Mit den bisher erlaeuterten Befehlen ist es bereits moeglich ein einfaches Pascal-Programm zu schreiben:

```
10 VAR EINGABE,ERGEBNIS:REAL;
20 (* KEINE UNTERPROGRAMME NOETIG *)
30 BEGIN (* ANFANG DES HAUPTPROGRAMMES *)
40 WRITELN ("ICH ERWARTE JETZT EINE ZAHL.");
50 READLN (EINGABE);
60 ERGEBNIS:= SIN(EINGABE);
70 WRITELN ("DER SINUS DER EINGABE IST:",ERGEBNIS);
80 END. (* ENDE DES HAUPTPROGRAMMES *)
```

Dieses Programm kann nun, wie in Kapitel 1 beschrieben, uebersetzt und gestartet werden.

4.2 Weiterfuehrende Ein- Ausgabemoeglichkeiten

Beim Pascal 64 gibt es die Moeglichkeit die Befehle WRITE, WRITELN, READ und READLN auf andere Geraete als Bildschirm und Tastatur zu beziehen. Dies geschieht wie in BASIC ueber die Files.

Um ein File eroeffnen zu koennen, muss man es zuerst im Variablenteil als Variable deklarieren:

VAR Name,Geraeteadr,Sekundaeradr,"Text,Name,etc.":FILE;

Diese Parameter entsprechen dem BASIC-Befehl OPEN, allerdings wird die Filenummer durch den Filenamen ersetzt. Das deklarierte File wird noch nicht eroeffnet.

Das Eroeffnen eines Files geschieht mit dem Befehl RESET:

RESET (Filename);

Der RESET-Befehl kann auch durch ein REWRITE ersetzt werden. Eine Unterscheidung zwischen Lese- und Schreibfile ist an dieser Stelle jedoch nicht noetig, dies geschah bereits bei der Deklaration des Files durch die Sekundaeradresse (Datasette) oder den angehaengten Text (Floppy).

Das Schliessen eines Files geschieht ueber den CLOSE-Befehl:

CLOSE (Filename);

Danach kann das File wieder eroeffnet werden. Ein File sollte immer vor Beendigung des Programmes geschlossen werden, da sonst Daten verloren gehen.

Der Zugriff auf ein File geschieht ueber die bereits erlaeuterten Ein- Ausgabebefehle (WRITE, WRITELN, READ, READLN), es wird nur als erstes Element der Variablenliste der Filename genannt. Um Daten mit READLN zu lesen, muessen diese ein entsprechendes Format haben, d.h. sie muessen mit WRITELN geschrieben worden sein.

Beispielprogramm:

```
10 VAR ZEICHEN:CHAR; FLOPPY,8,8,"TESTFILE,S,R":FILE;
20 DRUCKER,4,0,"":FILE; STATUS:INTEGER;
30 BEGIN
40 RESET (FLOPPY); RESET (DRUCKER);
50 REPEAT
60 READ (FLOPPY,ZEICHEN); STATUS:= PEEK(144); WRITE
(DRUCKER,ZEICHEN);
65 WRITE (ZEICHEN);
```

```
70 UNTIL STATUS=64; (* SCHLEIFENENDE *)
80 WRITELN; WRITELN ("FERTIG"); CLOSE (FLOPPY); CLOSE
(DRUCKER);
90 END.
```

Zum Verstaendnis der Schleife REPEAT ... UNTIL sehen Sie bitte in Kapitel 5 nach. Das Programm liest ein File, mit dem auf der Floppy gespeicherten Namen TESTFILE, von der Floppy und gibt es ueber Drucker und Bildschirm wieder aus. Durch Verwendung einer Variablen vom Typ CHAR, geschieht dies in einzelnen Bytes.

4.3 Der Filestatus

Die Funktion PEEK(144) entspricht dem Filestatus der letzten Ein- Ausgabeoperation und damit der BASIC-Variablen ST. Hat die Speicherstelle 144 den Inhalt 64, so ist das Fileende erreicht.

5. KAPITEL

5.1 Kontrollstrukturen

Um den Ablauf eines Programmes zu kontrollieren kann man Schleifen und bedingte Verzweigungen einsetzen. Einen Schleifentyp kennen Sie bereits von BASIC her: die FOR .. NEXT Schleife. Ausserdem kennen Sie mit dem IF .. THEN Befehl eine bedingte Verzweigung. Diese Verzweigung ist in BASIC aber zeilenbezogen, d.h. der Teil nach dem THEN ist durch das Zeilenende begrenzt. In Pascal wirkt ein Zeilenende dagegen nie als Begrenzung, sondern hoechstens als Leerzeichen. Um einen bedingt ausgefuehrten Block zu begrenzen, wird dieser in BEGIN und END; eingeschlossen, wie bereits in Kapitel 2 erlaeutert. Die Ausfuehrung eines Blockes kann nur bei diesen beiden Marken begonnen bzw. beendet werden. Bei dem Entwurf eines Programmes ist also schon die Blockstruktur zu beachten. Die Verwendung von Flussdiagrammen ist dazu wenig geeignet. Ein alternatives Verfahren, um ein Pascal-Programm grafisch darzustellen, wird in Kapitel 9 erlaeutert.
In BASIC gibt es noch die unbedingte Verzweigung, die durch den GOTO-Befehl realisiert ist. Aufgrund der leistungsfaehigen Strukturierbefehle des Pascal wurde dieser unstrukturierte, dem Pascal-Konzept widersprechende Befehl in Pascal 64 weggelassen. Einige Pascal-Versionen besitzen ihn zwar, aber auch dort ist er unnoetig und sollte moeglichst nicht benutzt werden.

5.2 Wiederholungsanweisungen (Schleifen)

5.2.1 Die REPEAT-Anweisung

Format:

REPEAT Befehlsblock UNTIL Bedingung;

REPEAT und UNTIL ersetzen dabei die Blockbegrenzer BEGIN und END.
Wirkung:

Der Block wird ausgefuehrt, dann die Bedingung (beliebige Vergleichsformel) ueberprueft und das ganze nochmal ausgefuehrt, wenn die Bedingung nicht erfuellt ist. Dies geschieht solange bis die Bedingung erfuellt ist. Danach wird die Programmbearbeitung nach dem UNTIL fortgesetzt.

Beispielprogramm:

```
10 VAR ZAEHLER:INTEGER;
20 BEGIN ZAEHLER:= 1;
30 REPEAT
40 WRITELN (ZAEHLER);
50 ZAEHLER:= ZAEHLER+1;
60 UNTIL ZAEHLER>10;
70 END.
```

Dieses Programm gibt die Zahlen von 1 bis 10 aus, was leicht nachvollziehbar ist. Die Zahl 11 wird nicht mehr ausgegeben, da vorher bereits die Bedingung ueberprueft und als erfuellt erkannt wurde. Zu beachten ist, dass die Variable ZAEHLER erst in einen definierten Ausgangszustand gebracht wird.

5.2.2 Die WHILE-Anweisung:

Format:

WHILE Bedingung DO Block;

Die Bedingung wird ueberprueft und der Befehlsblock ausgefuehrt, wenn diese erfuellt ist. Nach dem Ende des Blocks wird wieder bei WHILE begonnen. Erst wenn die Bedingung nicht erfuellt ist, wird mit dem Teil nach dem Block fortgefahren. Im Gegensatz zum UNTIL-Befehl ist es hier moeglich, dass der Block nicht ausgefuehrt wird, wenn die Bedingung nicht erfuellt ist.

Beispiel:

```
10 VAR ZAEHLER:INTEGER;
20 BEGIN ZAEHLER:= 1;
```

```
30 WHILE ZAEHLER<11 DO BEGIN
40 WRITELN (ZAEHLER);
50 ZAEHLER:= ZAEHLER+1;
60 END; (* ENDE DER SCHLEIFE *)
60 END.
```

Dieses Programm gibt ebenfalls die Zahlen 1 bis 10 aus. Ist ZAEHLER = 11, dann ist die Bedingung nicht mehr erfuellt und der Block wird uebersprungen.

5.2.3 Die FOR-Anweisung

Format:

FOR Variable:= Anfangswert TO Endwert DO Block;

Die Variable wird von Anfang bis Endwert in Einerschritten aufwaerts durchgezaehlt und jedesmal der Block ausgefuehrt. Ist der Anfangswert bei Start der Schleife groesser dem Endwert, wird der Block ueberhaupt nicht ausgefuehrt. Wie bei WHILE wird dann der Block uebersprungen. Nach Erreichen des Endwertes wird nach dem Block mit der Programmausfuehrung fortgefahren. Anfangs- und Endwert koennen beliebige Ausdruecke sein. Die Variable muss vom Typ INTEGER sein. Der Endwert wird vor jedem Blockdurchlauf neu berechnet, sein Wert kann also innerhalb des Blockes veraendert werden.
Man kann eine Schleife aber auch herunterzaehlen lassen:

FOR Variable:= Anfangswert DOWNTO Endwert DO Block;

Die Variable wird dabei in Einerschritten heruntergezaehlt und der Block uebersprungen, wenn der Anfangswert kleiner dem Endwert ist.

Beispiel:

```
10 VAR ZAEHLER:INTEGER; BEGIN
20 FOR ZAEHLER:= 1 TO 10 DO
30 BEGIN WRITELN (ZAEHLER); END;
40 END.
```

Widerum haben wir ein Programm, dass die Zahlen von 1 bis 10 ausdruckt. Durch das Verwenden einer FOR-Schleife wurde aber die Startwertzuweisung, das Heraufzaehlen der Variablen und die Ueberpruefung der Bedingung automatisiert. Um einen anderen Zaehlwert als +1 oder -1 zu verwenden, muss auf die WHILE-Schleife zurueckgegriffen werden. Bei der FOR-Schleife haetten wir die Blockbegrenzer BEGIN und END auch weglassen koennen, da der Block nur aus einem Befehl besteht.
Wie bereits erwaehnt koennen innerhalb eines Blockes beliebig weitere Strukturen verschachtelt werden. Die Verschachtelungstiefe ist dabei beliebig, muss aber dem Compiler angegeben werden. (siehe Kapitel 1)

Wie Sie gesehen haben gibt es in Pascal mehrere Moeglichkeiten ein Programm zu formulieren, welche die jeweils guenstigste ist haengt von dem jeweiligen Loesungsansatz ab. Sie sollten sich jedoch mit allen drei Schleifentypen vertraut machen. Mit REPEAT oder mit WHILE lassen sich nicht nur Zaehlschleifen darstellen. Folgendes Programm wartet z.B. auf das Druecken einer Taste:

```
10 BEGIN
20 REPEAT
30 UNTIL PEEK(203)<>64;
40 END.
```

Die Speicherstelle 203 enthaelt dabei einen Code fuer die momentan gedrueckte Taste. 64 steht fuer keine gedrueckte Taste. Ueberhaupt sind viele Speicheradressen zwischen 0 und 1023 in Pascal dieselben wie in BASIC. Natuerlich sind auch alle anderen Systemadressen erhalten geblieben (z.B. die Sprite-Register oder die Adresse des Bildschirmspeichers).

5.3Entscheidungsanweisungen

5.3.1 Die IF-Anweisung

Format:

IF Bedingung THEN Block1 ELSE Block2;

oder auch nur:

IF Bedingung THEN Block1

Wenn die Bedingung erfuellt ist, wird Block1 ausgefuehrt, andernfalls wird Block2 ausgefuehrt, soweit er vorhanden ist.

Beispiel:

```
10 VAR EINGABE:REAL;
20 BEGIN
30 READLN (EINGABE);
40 IF EINGABE<5 THEN BEGIN WRITELN ("DIE EINGABE IST  KLEINER
ALS 5"); END;
50 ELSE BEGIN WRITELN ("DIE EINGABE IST GROESSER ODER  GLEICH
5"); END;
60 END.
```

Auch in diesem Fall haetten wir BEGIN und END von Block1 und Block2 widerum weglassen koennen.
Dieses Programm macht nur eine grobe Aussage ueber die Variable EINGABE. Eine genauere Aussage ueber eine Variable oder einen Ausdruck ermoeglicht die folgende Anweisung:

5.3.2 Die CASE-Anweisung

Format:

```
CASE Ausdruck OF
Zahl,Zahl,Zahl,...,Zahl: Block1;
Zahl,...,Zahl: Block2
Zahl,...,Zahl: BlockN;
END;
```

Die Zahlen muessen vom Typ INTEGER sein, koennen aber auch durch Zeichen vom Typ CHAR (in Anfuehrungszeichen) ersetzt werden, wenn der Ausdruck ein entsprechendes Ergebnis liefert. Gleitkommazahlen sind nicht erlaubt.

Wirkungsweise:
Der Ausdruck wird ausgerechnet und der Block ausgefuehrt vor dem eine Zahl bzw. ein Zeichen steht, das mit dem Ergebnis identisch ist. Das END; beendet die Auflistung der Zahlen und Bloecke, es gehoert zum CASE-Statement und nicht zu einem Block. Ist keine der Zahlen gleich dem Ausdruck nach CASE, dann wird kein Block ausgefuehrt, sondern gleich nach dem END; des CASE-Statements fortgefahren.
Beispiel:

```
10 VAR EINGABE:INTEGER;
20 BEGIN
30 READLN (EINGABE);
40 CASE EINGABE OF
50 1,2,3: BEGIN WRITELN ("1 BIS 3"); END;
60 4,5,6: WRITELN ("4 BIS 6");
70 7,8,9,10,11: BEGIN WRITELN ("7 BIS 11"); END;
80 END; (* VON CASE *)
90 END. (* VOM PROGRAMM *)
```

Achten Sie bei der Programmeingabe darauf, dass Zeilennummer und Zahl durch ein geshiftetes Leerzeichen getrennt werden muessen (siehe Kapitel 1).
Dieses Programm gibt eine genauere Beschreibung der Variablen EINGABE, als das vorherige Programm dies tat.

Um die praktische Anwendung der Kontrollstrukturen zu verdeutlichen, folgt nun ein Programm zum Ausdruck der Primzahlen unter 2000. Das Programm arbeitet dabei nach dem Verfahren, das aus einer Tabelle alle Vielfachen einer Primzahl gestrichen werden, da Sie keine Primzahlen mehr sind. Wichtig ist nur, dass Sie die Verschachtelung der Bloecke erkennen.
Alle END-Statements wurden mit einer Bemerkung versehen, die angibt auf welche Struktur sie sich beziehen. Alle BEGIN und END, die weggelassen werden koennen, sind als Bemerkung dargestellt. Die Konstanten TRUE und FALSE dienen der besseren Lesbarkeit des Programms.

```
10 CONST N=1000; TRUE=-1; FALSE=0;
20 VAR Z:ARRAY(0..1000) OF INTEGER;
```

```
30 K,I:INTEGER;
40 BEGIN
50 FOR I:= 1 TO N DO
60     (* BEGIN *)
70     Z(I):= TRUE;
80     (* END VON FOR *)
90 WRITELN (2);
100 FOR I:= 1 TO N DO
110     (* BEGIN *)
120     IF Z(I) THEN
130        BEGIN
140        WRITELN (2*I+1);
150        K:= 1;
160        WHILE I+K*(2*I+1)<=N DO
170           BEGIN
180           Z(I+K*(2*I+1)):= FALSE;
190           K:= K+1;
200           END; (* VON WHILE *)
210        END; (* VON THEN *)
220     (* END VON FOR *)
230 END. (* DES PROGRAMMES *)
```

Tippen Sie nun dieses Programm ein, und uebersetzen Sie es wie in Kapitel 1 beschrieben. Es laeuft wesentlich schneller, als ein entsprechendes BASIC-Programm.
Fuegen Sie nun folgende Zeile ein:

```
45 INTEGER;
```

und aktivieren Sie den Compiler erneut. Die Geschwindigkeit des Programmes wurde stark gesteigert. Naeheres zu diesem Befehl erfahren Sie in Kapitel 7, vorerst sollten Sie ihn nicht benutzen.

6. KAPITEL

6.1 Unterprogramme und Funktionen

Die Unterprogrammdefinitionen stehen in Pascal noch vor dem Hauptprogrammblock und nach den Variablendeklarationen. Sie bestehen aus einem Parameterteil, den Unterprogrammvariablendeklarationen und dem Unterprogrammblock.

6.1.1 Unterprogrammdefinition

Format:

```
PROCEDURE Name (Parameterliste);
Variablendeklarationen
Block;
```

Die Parameterliste und die Variablendeklarationen koennen natuerlich auch fehlen. Die Variablendeklarationen wurden bereits in Kapitel 2 erlaeutert. Die Unterprogrammvariablen gehoeren nur zum Unterprogramm oder zu den vom Unterprogramm selber benutzten Unterprogrammen. Die Variablen des Hauptprogrammes koennen dagegen von allen Unterprogrammen und natuerlich vom Hauptprogramm benutzt werden. Sie werden deshalb globale Variablen genannt. Jeder Programmblock sollte nur seine eigenen Variablen benutzen, um Programmierfehler zu vermeiden. Der Datenaustausch zwischen Unterprogramm und aufrufendem Programm sollte sich deshalb moeglichst auf die Parameter beschraenken.
Mit der Parameterliste werden Variablen deklariert, denen beim Aufruf der PROCEDURE Werte oder Variablen des aufrufenden Programmes zugeordnet werden muessen. Es gibt dabei zwei Arten von Parametern:

Werteparameter

Diese Parameter werden wie Variablen deklariert, jedoch ohne das Wort VAR voranzustellen. Beim Aufruf des Unterprogrammes

wird dem Werteparameter vom aufrufenden Programm ein Wert uebergeben.

Variablenparameter:

Zur Unterscheidung von den Werteparametern wird das Wort VAR vor die Deklaration gestellt. Dem Variablenparameter wird beim PROCEDURE-Aufruf eine Variable uebergeben, mit der das Unterprogramm dann unter dem Namen des Variablenparameters arbeiten kann. So ist es moeglich, dass das Unterprogramm Variablen des aufrufenden Programmes benutzt, ohne dabei an eine bestimmte Variable gebunden zu sein. Jeder Aufruf der PROCEDURE kann eine andere Variable uebergeben.

Beispiel:

```
PROCEDURE TEST (VAR A,B:INTEGER; C:REAL; VAR X,Y:CHAR);
```

Die Variablen A, B, X und Y sind Variablenparameter, die Variable C ist ein Werteparameter. Alle Parameter koennen vom Unterprogramm wie Variablen benutzt werden.

6.1.2 Unterprogrammaufruf

Format:

```
PROCEDURE Name (Parameter1,Parameter2,...,ParameterN);
```

Dem Unterprogramm Name werden die Parameter uebergeben und dann das Unterprogramm ausgefuehrt. Die Parameteruebergabe geschieht in der Reihenfolge der Deklaration, Variablenparametern duerfen natuerlich nur Variablen uebergeben werden und keine Ausdruecke.
Der Aufruf unseres obigen Beispiels koennte also so lauten:

```
TEST (I,K,COS(3*5+Q),S,N);
```

Ein Unterprogramm kann natuerlich selber wieder Unterprogramme aufrufen, diese muessen aber bereits definiert sein. Ein PROCEDURE-Name darf nicht den Namen eines

Pascal-Befehls haben.

Beispiel:

```
10 VAR X:REAL;
20 PROCEDURE EINGABE (VAR ZAHL:REAL);
30 (* PROCEDUREVARIABLEN UNNOETIG *)
40 BEGIN
50 WRITELN ("BITTE ZAHL EINGEBEN");
60 READLN (ZAHL);
70 END; (* DER PROCEDURE *)
80 BEGIN (* DES PROGRAMMES *)
90 EINGABE (X);
100 WRITELN ("SIE HABEN ",X,"EINGEGEBEN");
110 END.
```

Das Unterprogramm wirkt wie die Definition eines neuen Befehles. Es waere z.B. moeglich den grafischen Befehlssatz des Pascal 64 zu erweitern (siehe Kapitel 8).

In vielen Faellen ist auch die Definition einer neuen Funktion wuenschenswert, auch dies ist in Pascal moeglich.

6.1.3 Definition von Funktionen

Format:

```
FUNCTION Name (Parameterliste):Funktionstyp;
Variablendeklarationen
Block;
```

Hier gelten die Vorschriften der PROCEDURE. Die Parameterliste muss aber mindestens ein Element enthalten. Ausserdem wird eine Funktionsvariable definiert, mit dem Namen der Funktion und vom Typ der im Funktionstyp angegeben ist. Dieser kann alle drei Datentypen darstellen.
Der Funktionsvariablen muss im Funktionsblock ein Wert zugewiesen werden, der dann das Ergebnis der Funktion repraesentiert. Ansonsten ist die Funktionsvariable eine normale Variable.

Der Aufruf einer Funktion geschieht wie bei den vordefinierten Funktionen, die Parameteruebergabe geschieht wie bei PROCEDUREN.

Beispiel:

```
10 VAR EINGABE:INTEGER;
20 FUNCTION FAKULTAET (N:INTEGER):REAL;
30 VAR ZAEHLER:INTEGER;
40 BEGIN
50 FAKULTAET:= 1;
60 FOR ZAEHLER:= 1 TO N DO
70 FAKULTAET:= FAKULTAET*ZAEHLER;
80 END; (* DER FUNKTION *)
90 BEGIN
100 READLN (EINGABE);
110 WRITELN (FAKULTAET(EINGABE));
120 END.
```

6.2 Rekursionen

In Pascal ist es moeglich, dass ein Unterprogramm oder eine Funktion sich selber aufruft, dies nennt man eine Rekursion. Eine Rekursion hat natuerlich nur dann einen Sinn, wenn die Folge der rekursiven Aufruefe irgendwann abbricht und sich wieder aufloest. Im Gegensatz zu BASIC werden dabei die Variablen des aufrufenden Programmes nicht zerstoert. Nach der Rueckkehr von einem Aufruf sind die Variablen wieder in dem Zustand in dem sie vor dem Aufruf waren. Da dies fuer alle Variablen des Unterprogrammes gilt, also auch fuer Felder, koennen komplexe Datenstrukturen aus einfachen Programmen entstehen. Variablenparameter sind in diesem Fall nicht einzetzbar, im Gegensatz zu anderen Pascal-Versionen. Die rekursive Uebergabe von Ergebnissen kann nur ueber die Funktionsvariable oder ueber globale Variablen geschehen (Variablen des Hauptprogrammes).

Beispiel:

```
10 VAR EINGABE:INTEGER;
20 FUNCTION FAKULTAET (N:INTEGER):REAL;
30 BEGIN
40 IF N>1 THEN FAKULTAET:= FAKULTAET(N-1)*N
50 ELSE FAKULTAET:= 1;
60 END;
70 BEGIN
80 READLN (EINGABE);
90 WRITELN (FAKULTAET(EINGABE));
100 END.
```

Da der Parameter N mit wachsender Anzahl der Aufruefe jeweils um 1 kleiner wird (N-1), erreicht er den Wert 1. Daraufhin sind die rekursiven Aufruefe beendet. Die Rekursion loest sich wieder auf, wobei alle N der verschiedenen Ebenen miteinander multipliziert werden. Somit gibt es auch einen rekursiven Weg, die Fakultaet zu ermitteln.

7. KAPITEL

7.1 Die Grafik-Befehle

7.1.1 Einschalten der Grafik

Das Umschalten vom Bildschirm auf die Grafik geschieht mit dem GRAPHIC-Befehl.

Format:

GRAPHIC Zahl;

Wenn die Zahl gleich 1 ist wird auf den Grafik-Bildschirm umgeschaltet, die Grafik dabei aber nicht geloescht. Die grafische Aufloesung betraegt 320 x 200 Punkte zweifarbig, die Hintergrundfarbe bleibt bestehen, die Punktfarbe ist gleich der Cursorfarbe. Eine mehrfarbige Grafik wird in Kapitel 8 realisiert.
Ist die Zahl gleich 0 dann wird auf den normalen Schirm zurueckgeschaltet und dieser geloescht.
Bei Anwendung des GRAPHIC-Befehls irgendwo im Programm wird der Bereich von 8192 bis 16384 fuer die Grafik reserviert, ausserdem wird der Bereich von 4100 bis 8191 fuer Maschinensprache frei. Diese beiden Bereiche koennen dann natuerlich auch anders genutzt werden.
Sprites koennen nicht im Bereich von 4096 bis 8191 liegen, in diesem Bereich greift der Videoprozessor auf den Zeichengenerator zu.

7.1.2 Loeschen der Grafik

Format:

SCREENCLEAR;

Dieser Befehl loescht den Grafikbildschirm.

7.1.3 Setzen eines Punktes

Format:

PLOT Ausdruck1,Ausdruck2;

PLOT setzt einen Punkt auf den Grafikbildschirm. Ausdruck1 gibt dabei die X-Koordinate (0-319), Ausdruck2 die Y-Koordinate (0-199) an.

7.1.4 Loeschen eines Punktes

Format:1n1
UNPLOT Ausdruck1,Ausdruck2;

Loescht einen Punkt und arbeitet ansonsten wie der PLOT-Befehl.
Befehle zum Zeichnen von Linien lassen sich leicht als Unterprogramme definieren (Kapitel 8).

Beispiel:

```
10 CONST PI=3.14159265; STEP=0.0196349541;
20 VAR X:INTEGER; I:REAL;
30 BEGIN GRAPHIC 1; SCREENCLEAR;
40 I:= PI*-1;
50 FOR X:= 0 TO 319 DO BEGIN
60 PLOT X,100+90*SIN(I);
70 I:= I+STEP;
80 END;
90 GRAPHIC 0; END.
```

7.1.5 Der SPRITE-Befehl

Format:

SPRITE Zahl,"Bitmuster";

Unter der durch die Zahl gegebenen Adresse und den folgenden Adressen wird das Bitmuster abgespeichert, welches Vielfache von 8 Bit enthalten kann. Zur Definition eines Sprites werden

dabei 24 Bit eingesetzt und der SPRITE-Befehl 21 mal benutzt, mit jeweils einer um 3 hoeheren Adresse. Leerzeichen repraesentieren geloeschte Bit, alle anderen Zeichen setzen ein Bit. Der SPRITE-Befehl kann natuerlich auch fuer andere Zwecke dienen, z.B. zum Erstellen eines neuen Zeichensatzes.

Beispiel:

```
SPRITE 960,"        XXXXXXXXXX        ";
SPRITE 963,"       XXXXXXXXXXXX       ";
```

Definiert die ersten beiden Zeilen des Sprites Nr. 15.
Zur Verfuegung stehen die Sprites 11 (Adresse 704 =64*11), 13 (832), 14 (896), 15 (960), 32 (2048), 33 (2112), 34 (2176), 35 (2240), 60 (3840), 61 (3904), 62 (3968) und 64 (4032).
Bei aktivierter Grafik sind ausserdem die Sprites 253 (16192), 254 (16256) und 255 (16320) zur Verfuegung.

7.2 Der POKE-Befehl

Format:

POKE Ausdruck1,Ausdruck2;

In die Speicherzelle die durch Ausdruck1 angegeben ist, wird das Ergebnis von Ausdruck2 geschrieben. Dies entspricht dem gleichnahmigen BASIC-Befehl.

7.3 Der SYS-Befehl

Format:

SYS Ausdruck1;

Ein Maschinenprogramm, das bei Ausdruck 1 beginnt, wird gestartet. Auch dieser Befehl ist mit dem BASIC-Befehl identisch. Fuer Maschinensprache ist der Bereich von $C000 bis $CFFF nutzbar, soweit keine rekursiven Aufruefe stattfinden, oder diese nur wenig Variablenspeicherplatz

benoetigen (rekursive Variablen werden ab $A000 aufwaerts gespeichert).

Beispiel:

SYS 64738;

setzt den Computer in den Einschaltzustand.

7.4 Der INTEGER; Befehl

Pascal 64 verfuegt ueber die Moeglichkeit arithmetische Operationen wahlweise im Ganzzahl oder Fliesskommaformat auszufuehren. Soweit das Ergebnis nicht davon beeintraechtigt wird, waehlt der Compiler das Ganzzahlformat. Bei den Verknuepfungen *, /, + und - ist dies nicht immer gegeben und kann vom Compiler nicht entschieden werden. Multipliziert man z.B. zwei INTEGER-Variablen, kann durchaus ein Ergebnis erscheinen, dass nicht von einer INTEGER-Variable aufgenommen werden kann. Darum gibt es in Pascal 64 den INTEGER; Befehl.

Format:

INTEGER;

Dieser Befehl bewirkt, dass die Operationen *, - und + im Ganzzahl-Format ausgefuehrt werden, wenn beide Operanden vom Typ INTEGER sind.
Fuer die ganzzahlige Division gibt es dagegen die Befehle DIV und MOD.
Unter Umstaenden koennen durch Bereichsueberlaeufe Rechenfehler entstehen, die bei Verwendung von Fliesskommarechnung nicht entstanden waeren. Deshalb ist der INTEGER; Befehl vorsichtig einzusetzen, besonders die Multiplikation ueberschreitet schnell den erlaubten Bereich von -32768 bis 32767.

7.5 Abschalten des Ganzzahlmodus

Format:

REAL;

Dieser Befehl schaltet wieder in den Ausgangszustand zurueck. Dies ist wichtig um z.B. innerhalb einer Schleife schnell rechnen zu koennen, aber ausserhalb dieser Schleife mit grossen Zahlen zu hantieren.
Beide Befehle wirken nur auf die folgenden Programmteile (also nicht auf Unterprogramme) und sind unabhaengig vom Verlauf des Programmes.

Beispiel:

```
10 BEGIN
20 IF 1=0 THEN INTEGER;
30 WRITELN (2*3);
40 END.
```

Die Multiplikation wird im Ganzzahlformat ausgerechnet, obwohl der Programmteil, in dem der INTEGER-Befehl enthalten ist, niemals durchlaufen wird.
Um z.B. das gesamte Programm im Ganzzahlmodus laufen zu lassen, ist der INTEGER; Befehl nach den BEGIN des ersten Unterprogrammes zu stellen.

7.6 Fehlermeldungen des uebersetzten Programmes

BAD SUPSCRIPT
Es wurde versucht eine Feldvariable anzusprechen, deren Index ausserhalb des deklarierten Bereiches liegt.

DIVISION BY ZERO
Eine Gleitkommavariable wurde durch Null dividiert.

ILLEGAL QUANTITY
Ein Wert liegt ausserhalb des fuer einen Befehl erlaubten Bereichs, z.B. bei unerlaubten Koordinaten der Befehle PLOT und UNPLOT.

OUT OF MEMORY

Der Maschinenstack oder der Rekursionstack ist uebergelaufen. Dies geschieht z.B. bei zu hoch verschachtelter Benutzung von selbstdefinierten Funktionen, aber auch durch zu hohe rekursive Unterprogrammverschachtelung.

OVERFLOW

Eine Gleitkommaberechnung hat den erlaubten Bereich (1.70141183E+38) ueberschritten.

Andere Fehlermeldungen entsprechen denen des BASIC-Interpreters. Syntax-Fehler werden schon vom Compiler gemeldet.

8. KAPITEL

Als Anregung zur Erstellung Ihrer eigenen Unterprogrammbibliothek folgen Unterprogramme zur besseren Handhabung der Grafik:

8.1 Zeichnen einer Linie

Der PROCEDURE DRAW muessen vier Parameter uebergeben werden, die ersten beiden Parameter bilden die Koordinaten des Anfangspunktes der Linie. Die letzten beiden Parameter geben die Endkoordinaten der Linie.

```
10 PROCEDURE DRAW (X1,Y1:REAL; X2,Y2:INTEGER);
20 VAR ZAEHLER,DX,DY:INTEGER; STEP:REAL;
30 BEGIN DX:= X2-X1; DY:= Y2-Y1;
40 IF ABS(DX)>ABS(DY) THEN BEGIN
50 STEP:= DY/ABS(DX);
60 FOR ZAEHLER:= X1 TO X2 DO BEGIN
70 PLOT ZAEHLER,Y1; Y1:= Y1+STEP; END; END;
80 ELSE BEGIN
90 STEP:= DX/ABS(DY);
100 FOR ZAEHLER:= Y1 TO Y2 DO BEGIN
110 PLOT X1,ZAEHLER; X1:= X1+STEP; END; END;
120 END;
```

Beim Einfuegen dieses Unterprogrammes in ein Programm muessen natuerlich die Zeilennummern entsprechend geaendert werden.

8.2Aktivieren einer vierfarbigen Grafik

Der PROCEDURE MULTICOLOR werden vier Parameter uebergeben, welche die Farben (0-15) der Vierfarbgrafik darstellen.

```
10 PROCEDURE MULTICOLOR (C0,C1,C2,C3:INTEGER);
20 VAR ZAEHLER,C12:INTEGER;
30 BEGIN INTEGER; GRAPHIC 1;
```

```
40 POKE 53270,PEEK(53270)OR16;
50 POKE 53281,C0;
60 C12:= C1*16+C2;
70 FOR ZAEHLER:= 1024 TO 2023 DO POKE ZAEHLER,C12;
80 FOR ZAEHLER:= 55296 TO 56295 DO POKE ZAEHLER,C3;
90 REAL; END;
```

8.3MULTIPLOT

Die Multiplot-Prozedur benoetigt drei Parameter. Dies sind X-Koordinate (0-159), Y-Koordinate (0-199) und Farbe (0-3) auf die aktivierten vier Farben bezogen.

```
10 PROCEDURE MULTIPLOT (XM,YM,C:INTEGER);
20 BEGIN INTEGER;
30 IF CAND1=1 THEN PLOT XM*2+1,YM;
40 ELSE UNPLOT XM*2+1,YM;
50 IF CAND2=2 THEN PLOT XM*2,YM;
60 ELSE UNPLOT XM*2,YM;
70 REAL; END;
```

Diese Routinen brauchen Sie nun nur noch in die Unterprogramm-Deklaration eines Pascal- Programmes einzufuegen, und koennen drei neue Befehle benutzen.

9. KAPITEL

9.1 Einfuehrung in das strukturiete Programmieren

Strukturiete Programmierung beruht auf der Idee ein Problem in verschiedene Bloecke aufzuteilen, die alle nur einen Eingang und einen Ausgang haben (BEGIN ... END), und die selbst wieder aus Bloecken zusammengesetzt sind. Bloecke koennen nur hintereinander, nebeneinander (bei bedingten Verzweigungen) oder in Schleifen stehen, sie duerfen nicht ueberlappen und es duerfen keine Spruenge erfolgen. Durch Darstellen eines Blocks als viereckigen Kasten lassen sich sogenannte Struktogramme erstellen, die eine gute grafische Alternative zu den Flussdiagrammen sind. Innerhalb eines Blockes wird dann seine Struktur erlaeutert (Schleife oder Verzweigung) und seine Teilbloecke in ihn hineingezeichnet. Bei bedingten Verzweigungen wird ueber jeden Block die Bedingung geschrieben, die zu seiner Ausfuehrung erfuellt sein muss.
Um zu einer solchen Grafik zu kommen, wird das zu loesende Problem in einzelne Teilprobleme (Bloecke) zerlegt. Diese Teilprobleme werden schrittweise weiter zerlegt bis die einzelnen Verfeinerungstufen so klar definiert sind, dass sie problemlos in Pascal dargestellt werden koennen. Dabei sollte die kleinste Einheit maximal eine Bildschirmseite umfassen, so dass mit Hilfe schriftlicher Aufzeichnungen stets der Ueberblick behalten wird.
Ueber jedes Teilproblem (Block) sollten schriftlich festgehalten werden:

- Die gegebenen Variablen, Felder, Parameter, Files etc.

- Was der Block mit diesen Daten macht.

- Wie die Ergebnisse an weitere Bloecke uebergeben werden. (Variablennamen)

- Ob Daten an den Benutzer oder Ausgabegeraete uebergeben werden.

Ein kleines praktisches Beispiel waere z.B. folgendes Problem:
Die Zahlen von 1 bis 1000 sollen auf Primzahlen durchsucht werden und alle Primzahlen ausgedruckt werden.
Dieses Problem laesst sich aufspalten in:
- Die Ueberpruefung einer Zahl, ob sie eine Primzahl ist.
- Die Erzeugung der Zahlen von 1 bis 1000 und die Uebergabe an den Ueberpruefungsblock.

Die Zahl 1 braucht dabei nicht ueberprueft werden. Die zu ueberpruefende Zahl nimmt die Variable N auf.
Die Grobstruktur des Programmes sehen dann so aus:

```
10 VAR N:INTEGER; BEGIN
20 FOR N:= 2 TO 1000 DO BEGIN
30 Ueberpruefung auf Primzahl
60 Ausdrucken, wenn Primzahl
70 END; END.
```

Um festzustellen, ob eine Zahl eine Primzahl ist, wird versucht diese durch alle Zahlen bis zu ihrer Wurzel zu teilen.
Dazu muessen zuerst diese Zahlen generiert werden. Dazu wird die Variable ZAEHLER benutzt:

```
10 VAR N,ZAEHLER:INTEGER; BEGIN
20 FOR N:= 2 TO 1000 DO BEGIN ZAEHLER:= 1;
30 REPEAT
40 ZAEHLER:= ZAEHLER+1;
50 UNTIL ZAEHLER>SQRT(N);
70 END; END.
```

Nun braucht nur noch auf Teilbarkeit untersucht, und die Schleife entsprechend abgebrochen werden:

```
10 VAR N,ZAEHLER:INTEGER; BEGIN
20 FOR N:= 2 TO 1000 DO BEGIN ZAEHLER:= 1;
30 REPEAT
40 ZAEHLER:= ZAEHLER+1;
50 UNTIL (ZAEHLER>SQRT(N))OR(NMODZAEHLER=0);
60 IF NOT(NMODZAEHLER=0) THEN WRITELN (N);
70 END; END.
```

Die Zahl ist eine Primzahl, wenn alle Teilungen einen Rest besitzen. Die Restberechnung geschieht durch MOD.
Dieses Programm kann nun noch nach belieben optimiert werden, z.B. ist die Ueberpruefung von geraden Zahlen unnoetig. Bei groesseren Programmen sollte durch passende Wahl der Variablennamen und durch Bemerkungen (* Bemerkung *) die Lesbarkeit erhoeht werden.
Ein ausgefeiltes Programm zur Primzahlberechnung ist in Kapitel 5 enthalten, es benutzt allerdings eine andere Methode.

10. KAPITEL

10.1Pascal 64 Kurzuebersicht

10.1.1 Datentypen

REAL - Bereich +-1.70141183E+38, 9 Stellen Rechengenauigkeit
INTEGER - Bereich +- 32767, keine Nachkommastellen
CHAR - Alle darstellbaren Zeichen, Laenge 1 Zeichen

10.1.2 Deklaration

VAR Variablenliste:Typ;
-Variablendeklaration

VAR Liste:ARRAY(0..Zahl) OF TYP;
-Felderdeklaration

CONST Name=Wert;
-Konstantendeklaration

10.1.3 Unterprogramme:

PROCEDURE Name (Parameterliste); Variablendeklaration Block;
Definition eines Unterprogrammes, dass wie ein Befehl aufgerufen wird.

FUNCTION Name (Parameterliste):Typ Variablendeklarationen Block;
Definition eines Unterprogrammes, dass wie eine Funktion aufgerufen wird.

10.1.4 Strukturen:

REPEAT Block; UNTIL Bedingung;
Wiederholt den Block, bis die Bedingung erfuellt ist.

WHILE Bedingung DO Block;
Wiederholt den Block, solange die Bedingung erfuellt ist.

FOR Variable:= Anfangswert TO Endwert DO Block;
Zaehlt die Variable von Anfangswert bis Endwert hoch und fuehrt jedesmal den Block aus.

FOR Variable:= Anfangswert DOWNTO Endwert DO Block;
Zaehlt die Variable von Anfangswert bis Endwert hinunter und fuehrt jedesmal den Block aus.

IF Bedingung THEN Block;
Wenn die Bedingung erfuellt ist, wird der Block ausgefuehrt.

IF Bedingung THEN Block1; ELSE Block2;
Wenn die Bedingung erfuellt ist, wird Block1 ausgefuehrt, andernfalls Block2.

```
CASE Ausdruck OF
Werteliste: Block;
Werteliste: Block;
  ....        ...
END;
```
Wenn ein Wert mit dem Ergebnis des Ausdrucks identisch ist, wird der dahinterstehende Block ausgefuehrt.

10.1.5 Ein- und Ausgabebefehle

READ (Variablenliste);
Die Variablen werden von der Tastatur eingelesen.

READLN (Variablenliste);
Die Variablen werden von der Tastatur eingelesen und ein 'RETURN' wird erwartet.

WRITE (Ausdruck,"Text",...);
Die Ergebnisse der Ausdruecke und die Texte werden auf den Bildschirm ausgegeben.

WRITELN (Ausdruck,"Text",...);
Wie WRITE, zusaetzlich wird ein 'RETURN' ausgegeben.

VAR Name,Geraet,Sekundaeradresse,"Text":FILE;
Ein File wird definiert.

RESET (Name); oder REWRITE (Name);
Ein File wird eroeffnet.

CLOSE (Name);
Ein File wird geschlossen.

WRITE (Filename,Liste); oder WRITELN (Filename,Liste);
Umlegen einer Ausgabe auf ein File.

READ (Filename,Liste); oder READLN (Filename,Liste);
Umlegen einer Eingabe auf ein File.

10.1.6 Sonstige Befehle

GRAPHIC 1;
Einschalten einer hochaufloesenden Grafik.

GRAPHIC 0;
Abschalten einer Grafik.

SCREENCLEAR;
Loeschen des Grafikbildschirms.

PLOT Ausdruck1,Ausdruck2;
Setzen eines grafischen Punktes.

UNPLOT Ausdruck1,Ausdruck2;
Loeschen eines grafischen Punktes.

SPRITE Adresse,"Bitmuster";
Schreiben eines Bitmusters in einen Speicherbereich.

POKE Ausdruck1,Ausdruck2;
In die Speicherzelle mit der Adresse von Ausdruck1 wird

Ausdruck2 geschrieben.

SYS Ausdruck;
Startet ein Maschinenprogramm.

INTEGER;
Einschalten des Ganzzahlrechenmodus.

REAL;
Zurueckschalten auf den normalen Rechenmodus;

10.1.7 Zuordnung

Variable:= Ausdruck;
In die Variable wird das Ergebnis des Ausdrucks geschrieben.

10.1.8 Rechenoperationen

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
DIV	Ganzzahldivision
MOD	Berechnung des Restes
AND	logisches und binaeres "und"
OR	logisches und binaeres "oder"
=	gleich
<>	ungleich
>	groesser
<	kleiner
<=	kleiner oder gleich
>=	groesser oder gleich

10.1.9 Funktionen

SIN	Sinus, Winkel in Bogenmass
COS	Cosinus
TAN	Tangens

ARCTAN	Arcustangens
EXP	Potensierung mit e
LN	natuerlicher Logarithmus
SQR	Quadrat
SQRT	Quadratwurzel
ABS	Betrag einer Zahl
TRUNC	ganzahliger Anteil
NOT	logisches und binaeres "nicht"
PEEK	Inhalt einer Speicherstelle
ORD	Zahlencode eines Zeichens
CHR	Zeichen fuer einen Code

DIE BEDIENUNG VON PASCAL 64

1. Hinweis

Beachten Sie, daß die PASCAL 64-Diskette kopiergeschützt ist. Das Directory dieser Diskette wird nicht angezeigt! Aus diesem Grunde sollten Sie Ihre Benutzerprogramme auf eine andere Diskette (Arbeitsdiskette) ablegen. Zum Erzeugen eines ablauffähigen Programmes ist unbedingt der LOADER der PASCAL 64-Diskette erforderlich. Ein mit dem LOADER abgespeichertes, endgültiges Objektprogramm kann nicht mit GOTO 100 zum Erzeugen ablauffähiger Programme benutzt werden.

2. Erstellen von PASCAL-Programmen

- Laden Sie den PASCAL 64-Compiler mit LOAD "COMPILER",8. Dieser Compiler belegt die BASIC-Zeilen ab 10000.
- Geben Sie Ihr PASCAL-Programm innerhalb der Zeilen 1-9999 ein
- Legen Sie Ihre Arbeitsdiskette ein, auf der das übersetzte Programm gespeichert werden soll.
- Starten Sie den Compiler mit RUN. Geben Sie den Programmnamen, den Massenspeicher (D - Diskette oder K - Kassette) und die Verschachtelungstiefe an.
- Sollte auf der Diskette bereits ein File mit diesem Programmnamen enthalten sein, so blinkt die Leuchtdiode der Floppy. Unterbrechen Sie dann den Compiler mit der Taste RUN/STOP und Starten ihn erneut mit RUN. Geben Sie nun einen anderen Programmnamen an.
- Der Compiler gibt die Anzahl der noch freien Bytes und das Programmende an. Dieses Programmende muß später beim Lauf des LOADER angegeben werden.
- Das Objektprogramm kann nun gemeinsam mit dem Compiler auf Ihre Arbeitsdiskette gespeichert werden. Benutzen Sie dazu den Befehl SAVE "name",8.
- Der vom Compiler erzeugte Zwischencode befindet sich nach der Übersetzung auf der Arbeitsdiskette und kann mit dem LOADER in ein lauffähiges Programm umgewandelt werden. Laden Sie dazu den LOADER von der PASCAL 64-Systemdiskette mit LOAD "LOADER",8 und Starten ihn mit GOTO 100. Danach entnehmen Sie die PASCAL 64-Diskette wieder aus dem Laufwerk und legen Ihre Arbeitsdiskette ein. Geben Sie nun das Programmende und den Programmnamen ein.
- Sollte der LOADER den angegebenen Programmnamen auf der Diskette nicht finden, so blinkt die Leuchtdiode der Floppy. Unterbrechen Sie in diesem Fall den LOADER mit der Taste RUN/STOP, legen die PASCAL 64-Systemdiskette wieder ein, Starten den LOADER mit GOTO 100 und wechseln die Systemdiskette gegen die Arbeitsdiskette aus.
- Nun wird die Meldung "READY" ausgegeben und das Programm kann mit RUN gestartet werden.
- Dieses endgültige, ablauffähige Programm kann nun zusammmen mit dem LOADER auf einer Diskette gespeichert werden. Dieser LOADER kann nicht zur Übersetzung des Compiler-Codes benutzt werden. Er dient lediglich zum Ablauf des fertigen Programmes.

Beachten Sie unbedingt, daß Ihre Programme nicht auf der PASCAL 64-Diskette gespeichert werden.